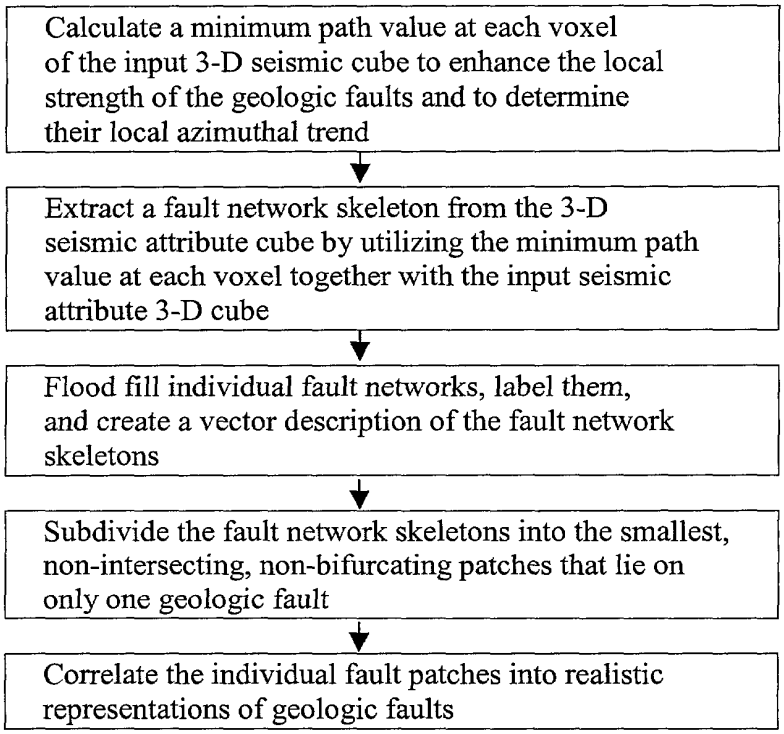
FIG. 1


Calculate a minimum path value at each voxel of the input 3-D seismic cube to enhance the local strength of the geologic faults and to determine their local azimuthal trend
Extract a fault network skeleton from the 3-D seismic attribute cube by utilizing the minimum path value at each voxel together with the input seismic attribute 3-D cube
Flood fill individual fault networks, label them, and create a vector description of the fault network skeletons
Subdivide the fault network skeletons into the smallest, non-intersecting, non-bifurcating patches that lie on only one geologic fault
Correlate the individual fault patches into realistic representations of geologic faults

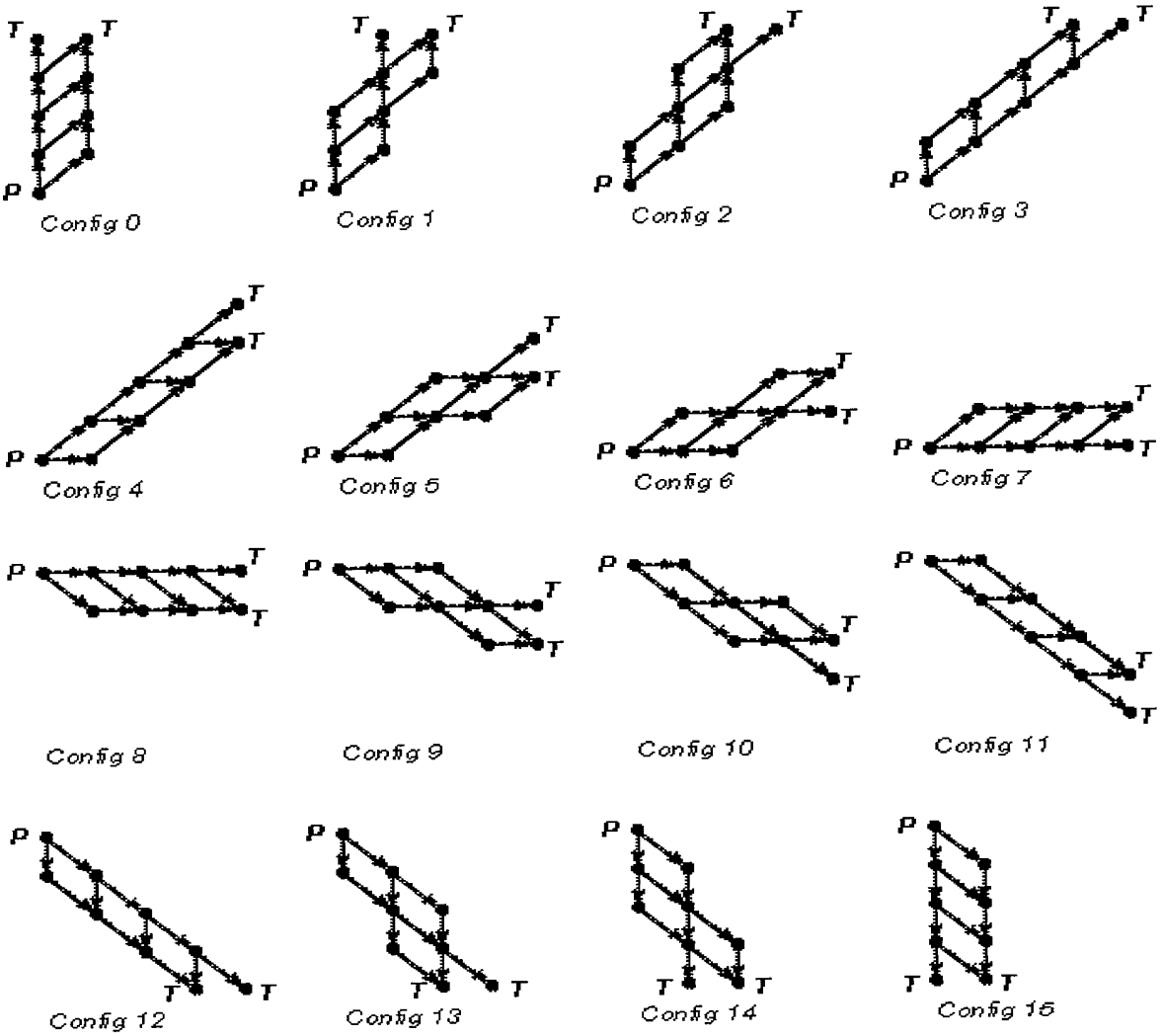

Config 0
Config 1
Config 2
Config 3
Config 4
Config 5
Config 6
Config 7
Config 8
Config 9
Config 10
Config 11
Config 12
Config 13
Config 14
Config 15

FIG. 2

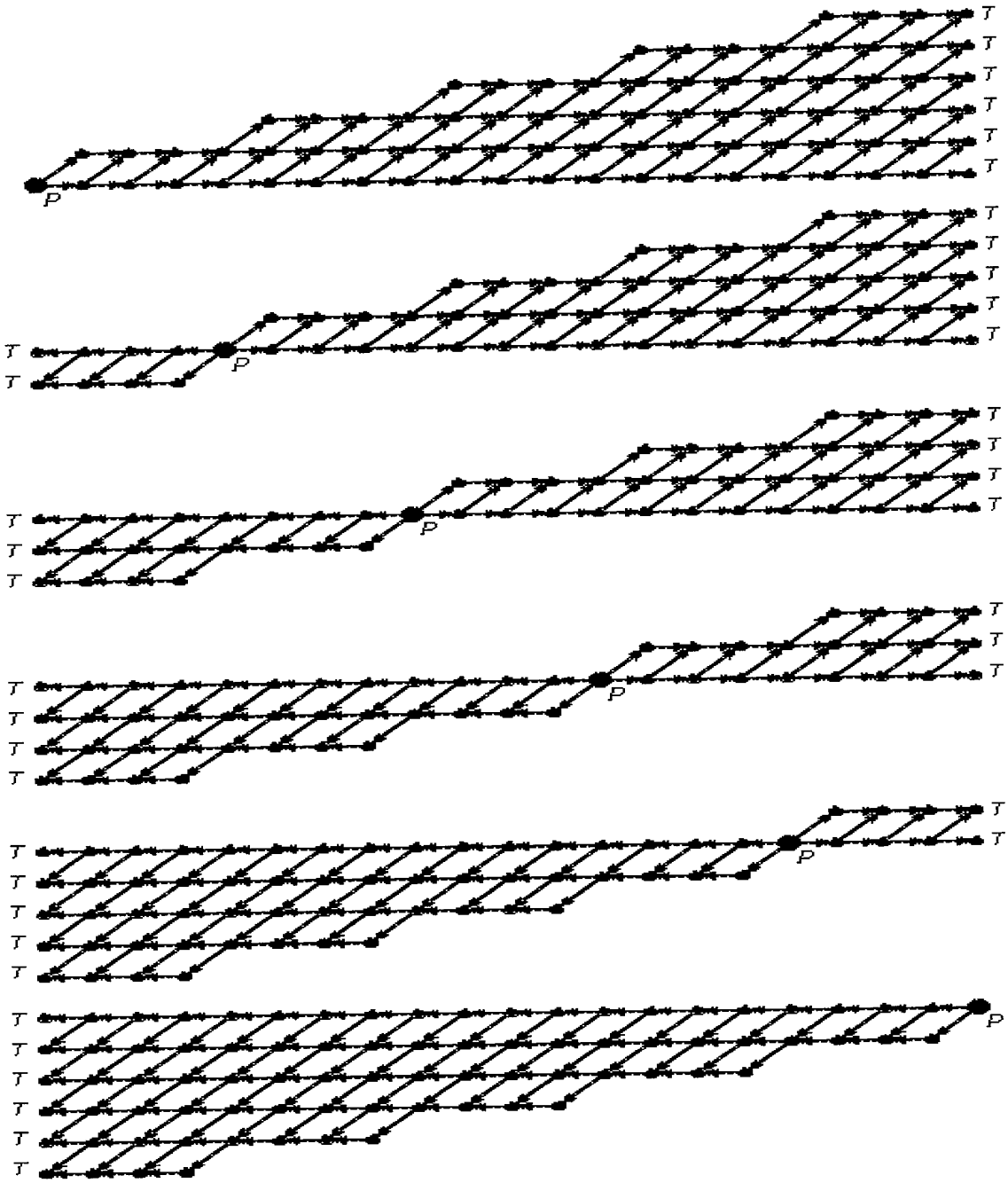


FIG. 3

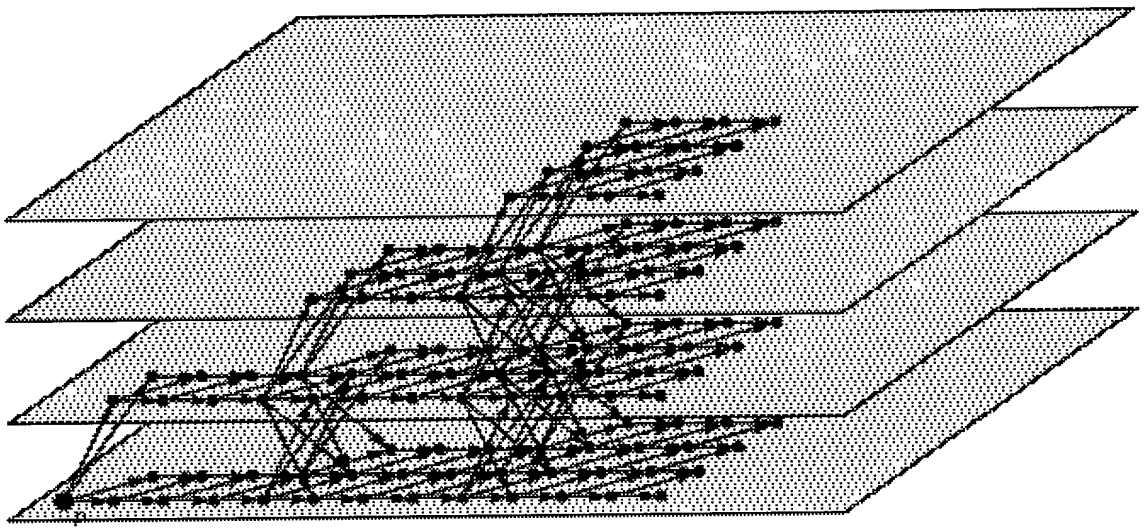
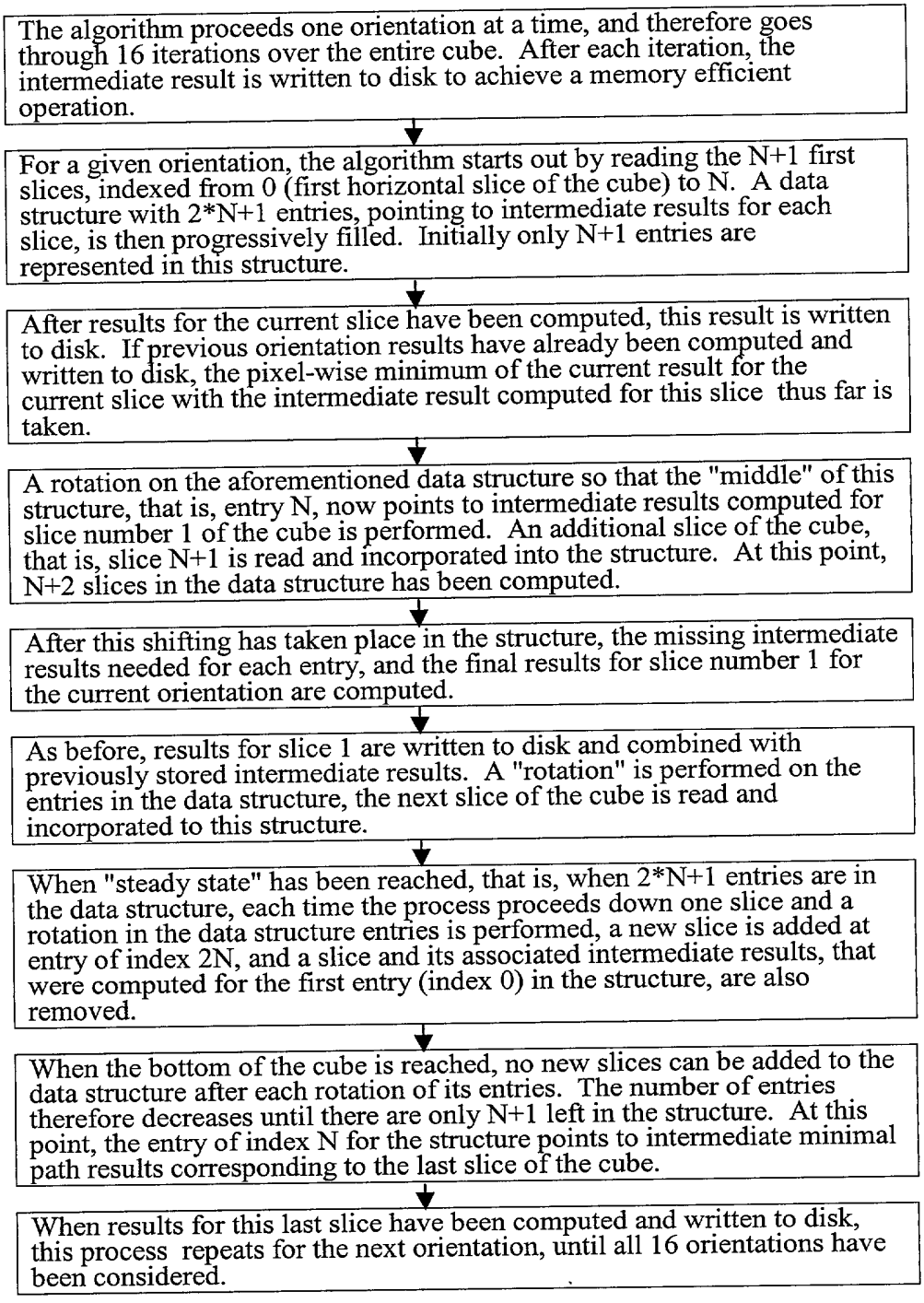


FIG. 4

FIG. 5


The algorithm proceeds one orientation at a time, and therefore goes through 16 iterations over the entire cube. After each iteration, the intermediate result is written to disk to achieve a memory efficient operation.
For a given orientation, the algorithm starts out by reading the N+1 first slices, indexed from 0 (first horizontal slice of the cube) to N. A data structure with 2*N+1 entries, pointing to intermediate results for each slice, is then progressively filled. Initially only N+1 entries are represented in this structure.
After results for the current slice have been computed, this result is written to disk. If previous orientation results have already been computed and written to disk, the pixel-wise minimum of the current result for the current slice with the intermediate result computed for this slice thus far is taken.
A rotation on the aforementioned data structure so that the "middle" of this structure, that is, entry N, now points to intermediate results computed for slice number 1 of the cube is performed. An additional slice of the cube, that is, slice N+1 is read and incorporated into the structure. At this point, N+2 slices in the data structure has been computed.
After this shifting has taken place in the structure, the missing intermediate results needed for each entry, and the final results for slice number 1 for the current orientation are computed.
As before, results for slice 1 are written to disk and combined with previously stored intermediate results. A "rotation" is performed on the entries in the data structure, the next slice of the cube is read and incorporated to this structure.
When "steady state" has been reached, that is, when 2*N+1 entries are in the data structure, each time the process proceeds down one slice and a rotation in the data structure entries is performed, a new slice is added at entry of index 2N, and a slice and its associated intermediate results, that were computed for the first entry (index 0) in the structure, are also removed.
When the bottom of the cube is reached, no new slices can be added to the data structure after each rotation of its entries. The number of entries therefore decreases until there are only N+1 left in the structure. At this point, the entry of index N for the structure points to intermediate minimal path results corresponding to the last slice of the cube.
When results for this last slice have been computed and written to disk, this process repeats for the next orientation, until all 16 orientations have been considered.

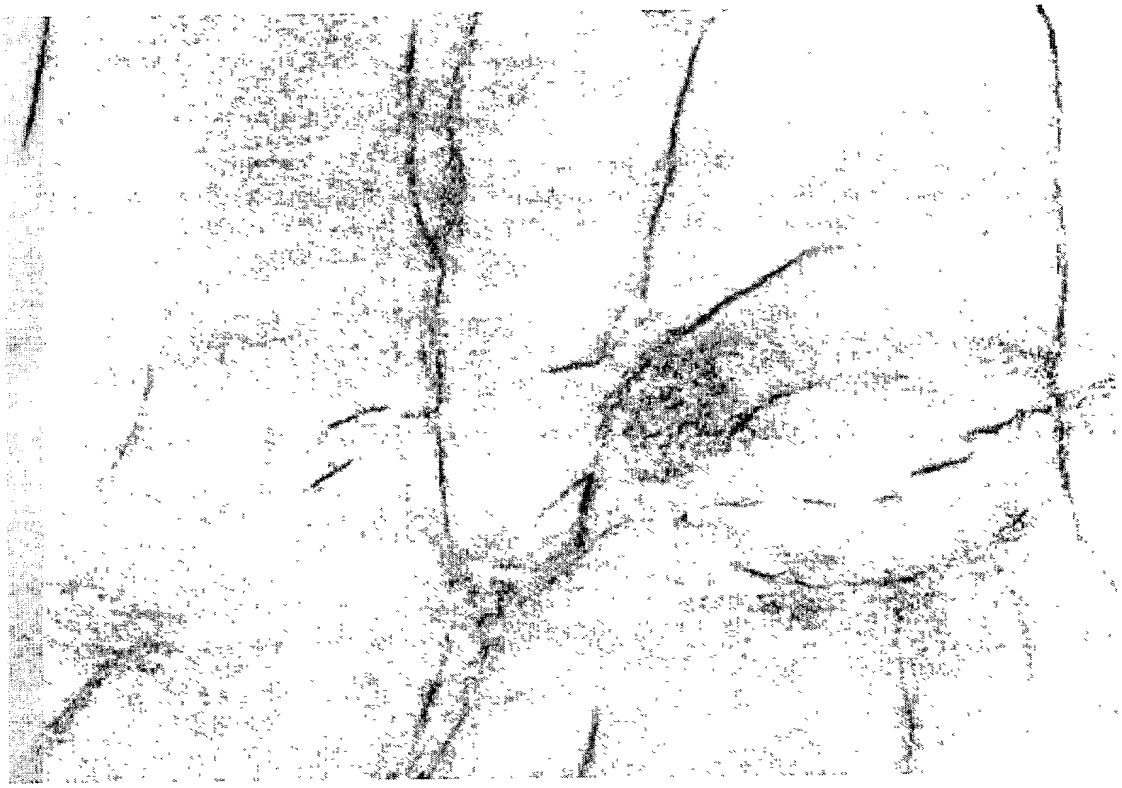


FIG. 6

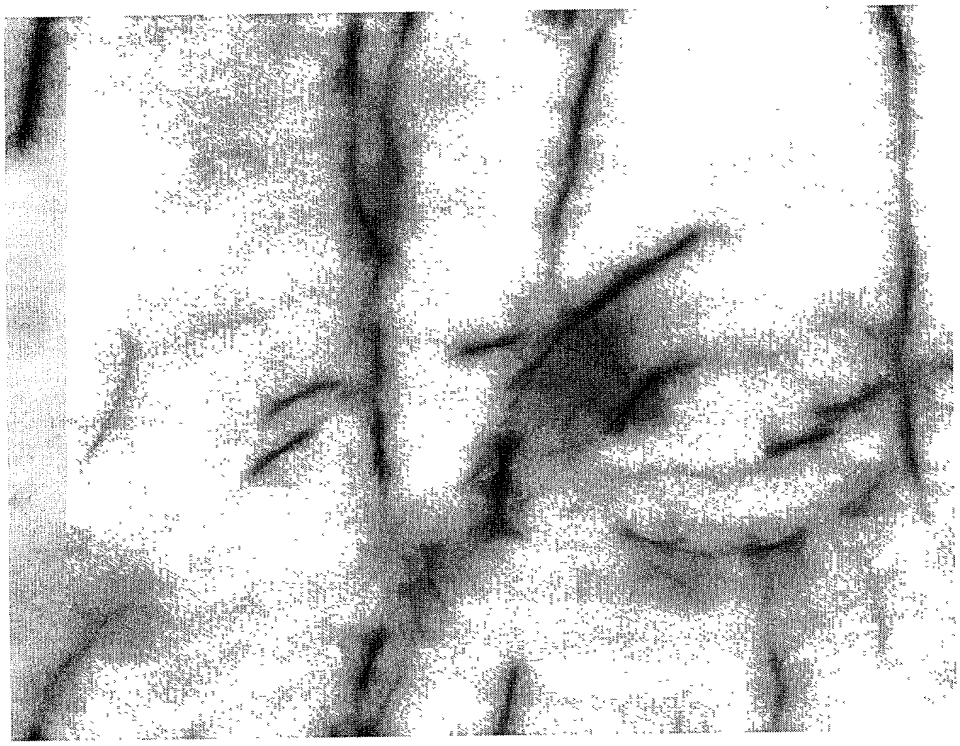


FIG. 7

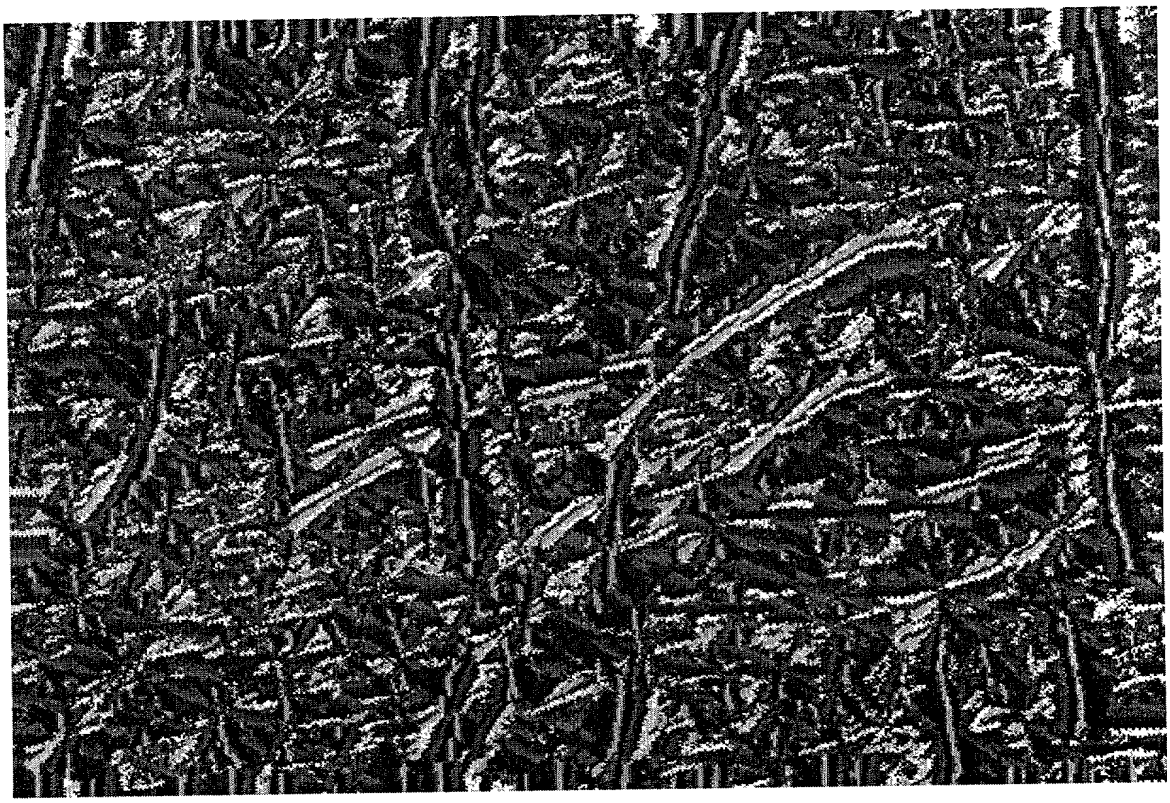
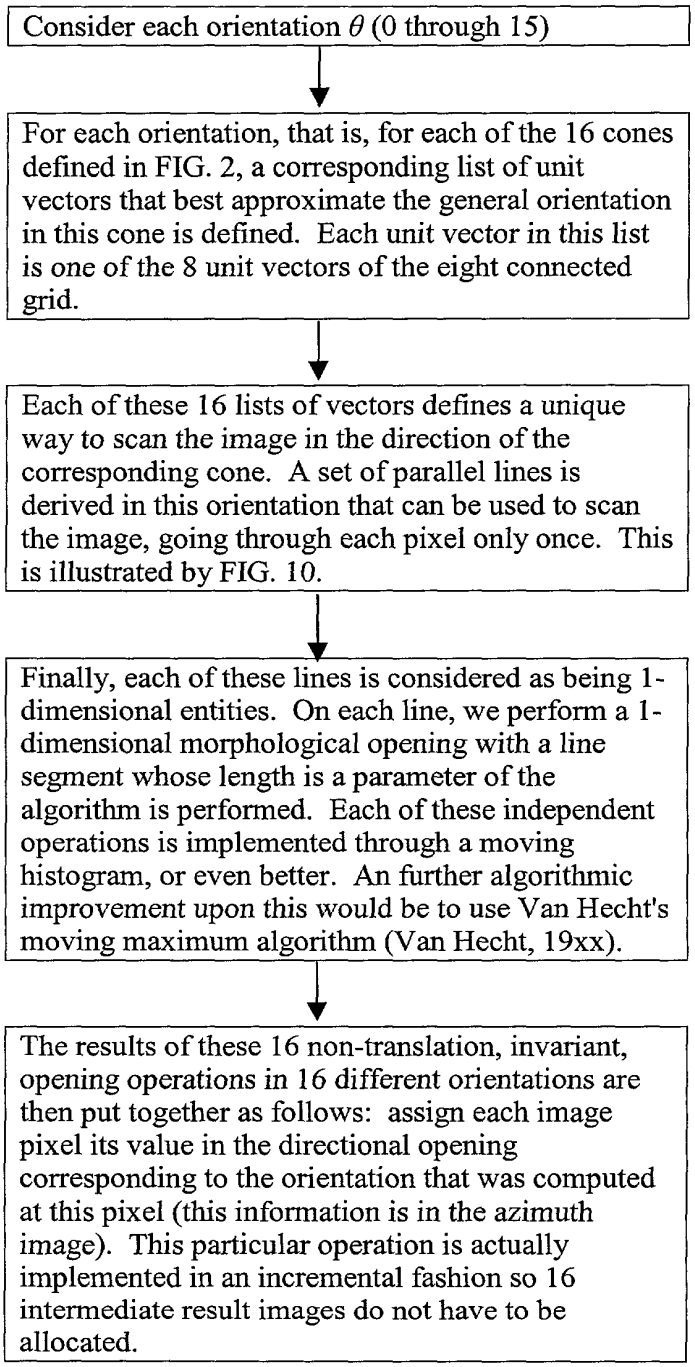


FIG. 8

FIG. 9


Consider each orientation θ (0 through 15)
For each orientation, that is, for each of the 16 cones defined in FIG. 2, a corresponding list of unit vectors that best approximate the general orientation in this cone is defined. Each unit vector in this list is one of the 8 unit vectors of the eight connected grid.
Each of these 16 lists of vectors defines a unique way to scan the image in the direction of the corresponding cone. A set of parallel lines is derived in this orientation that can be used to scan the image, going through each pixel only once. This is illustrated by FIG. 10.
Finally, each of these lines is considered as being 1-dimensional entities. On each line, we perform a 1-dimensional morphological opening with a line segment whose length is a parameter of the algorithm is performed. Each of these independent operations is implemented through a moving histogram, or even better. An further algorithmic improvement upon this would be to use Van Hecht's moving maximum algorithm (Van Hecht, 19xx).
The results of these 16 non-translation, invariant, opening operations in 16 different orientations are then put together as follows: assign each image pixel its value in the directional opening corresponding to the orientation that was computed at this pixel (this information is in the azimuth image). This particular operation is actually implemented in an incremental fashion so 16 intermediate result images do not have to be allocated.

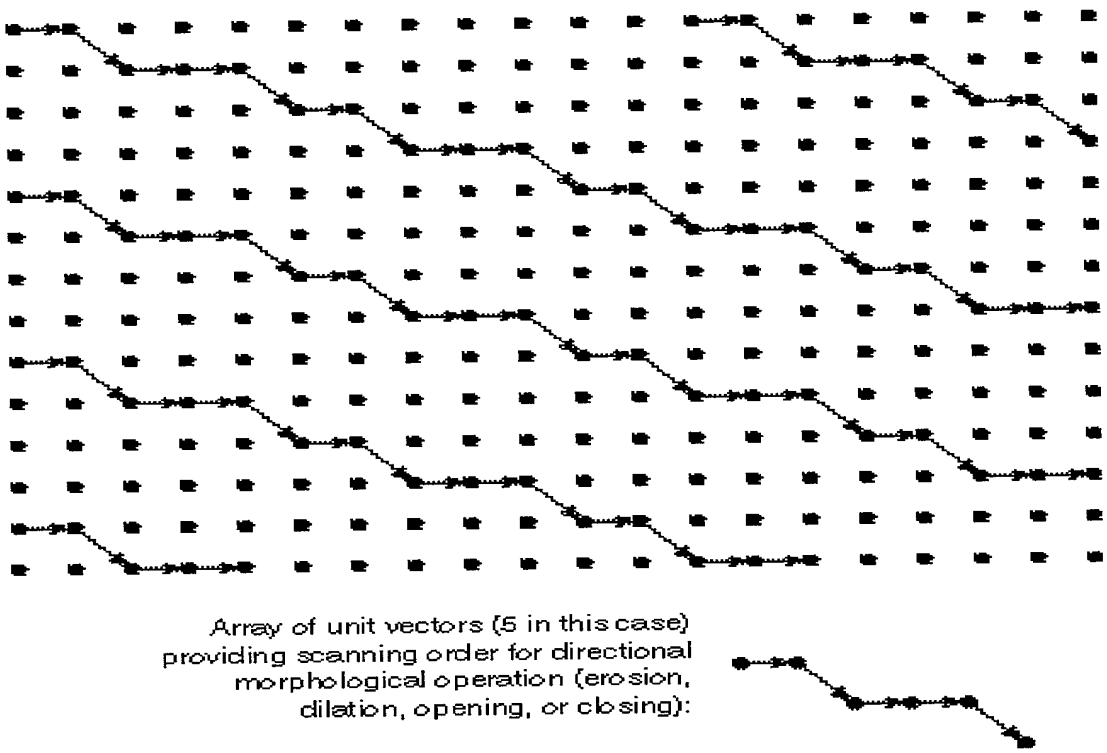

Array of unit vectors (5 in this case) providing scanning order for directional morphological operation (erosion, dilation, opening, or closing):

FIG. 10

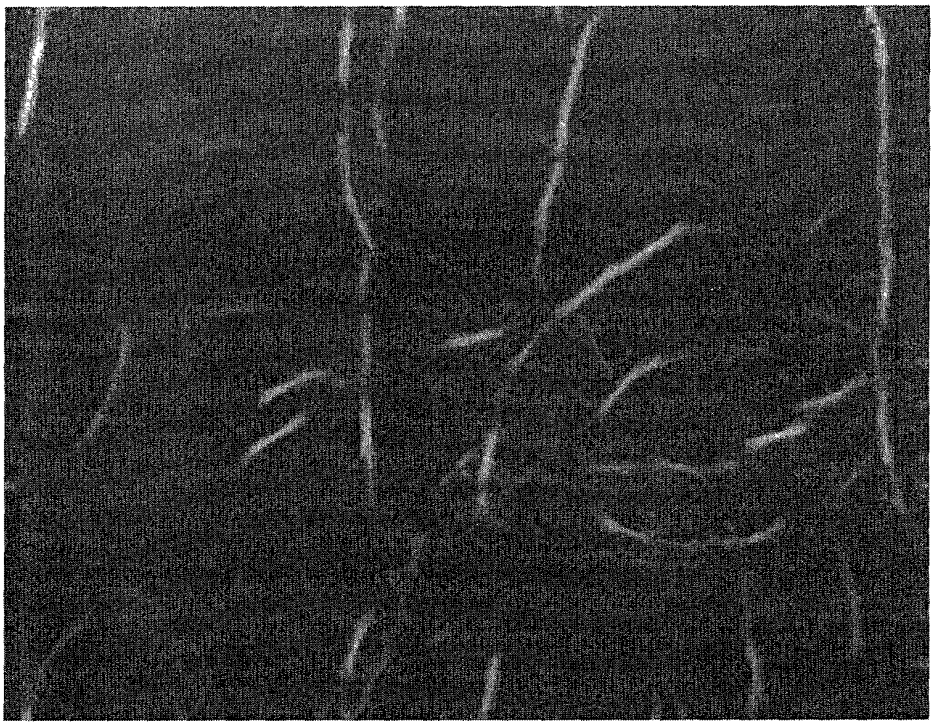


FIG. 11

FIG. 12

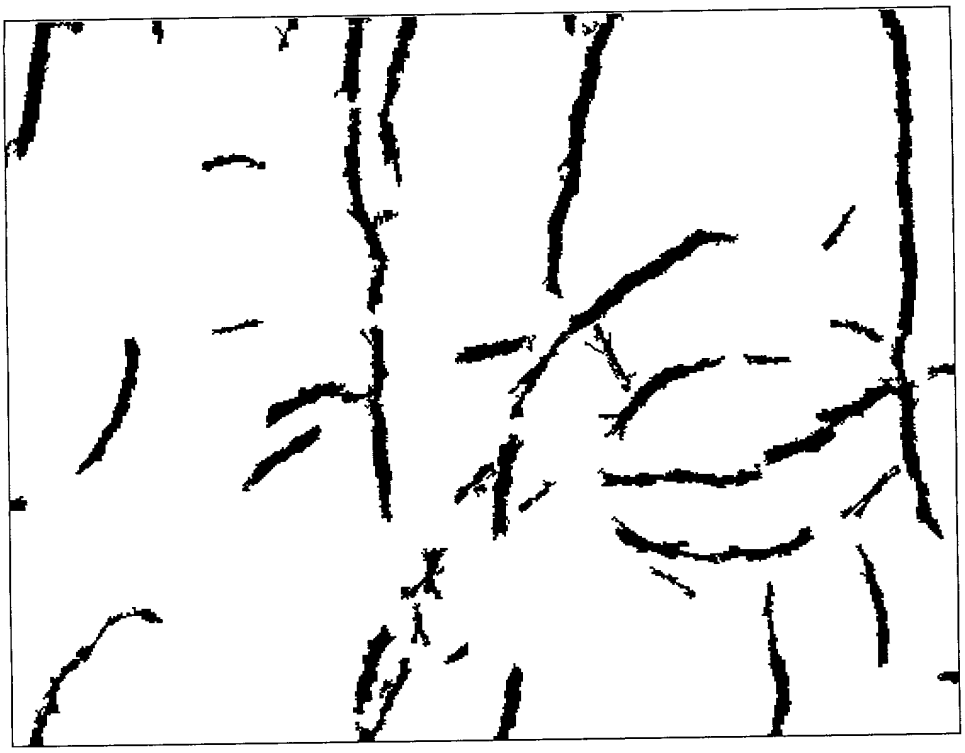


FIG. 13

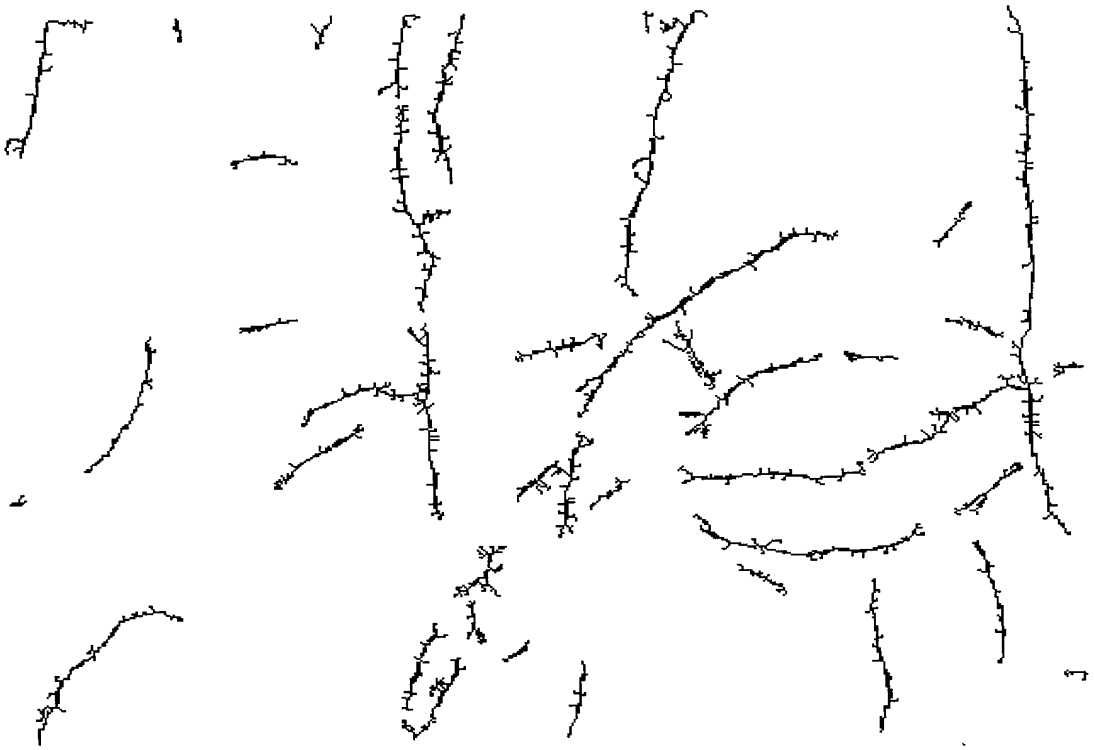


FIG. 14

FIG. 15

FIG. 16

FIG. 17

FIG. 18

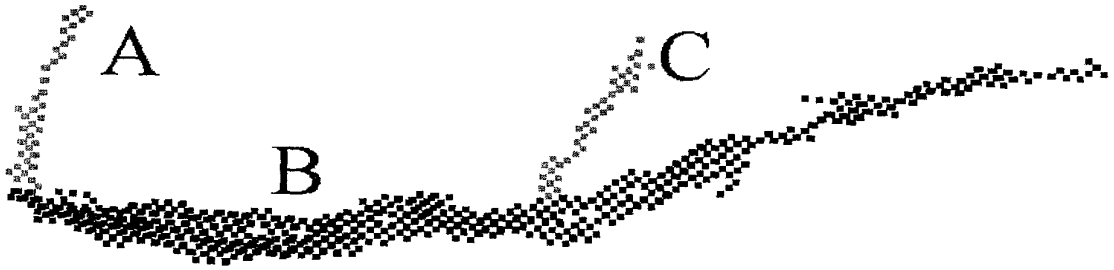


FIG. 19

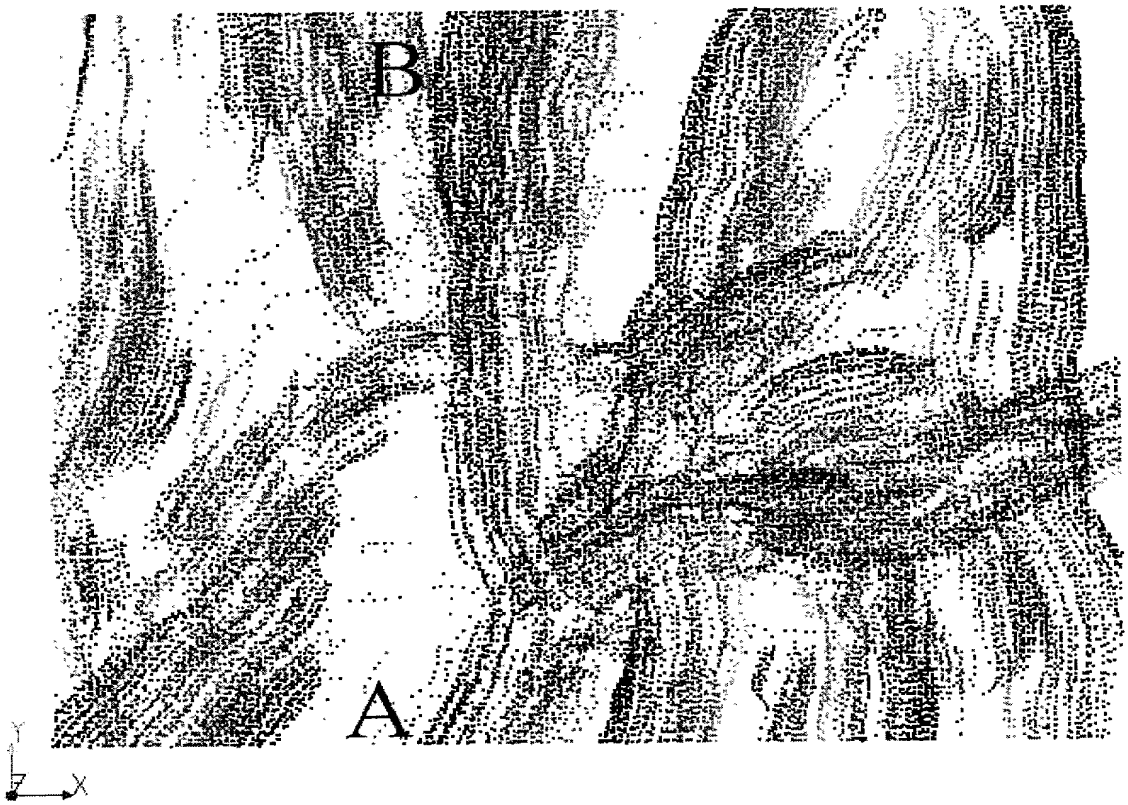


FIG. 20

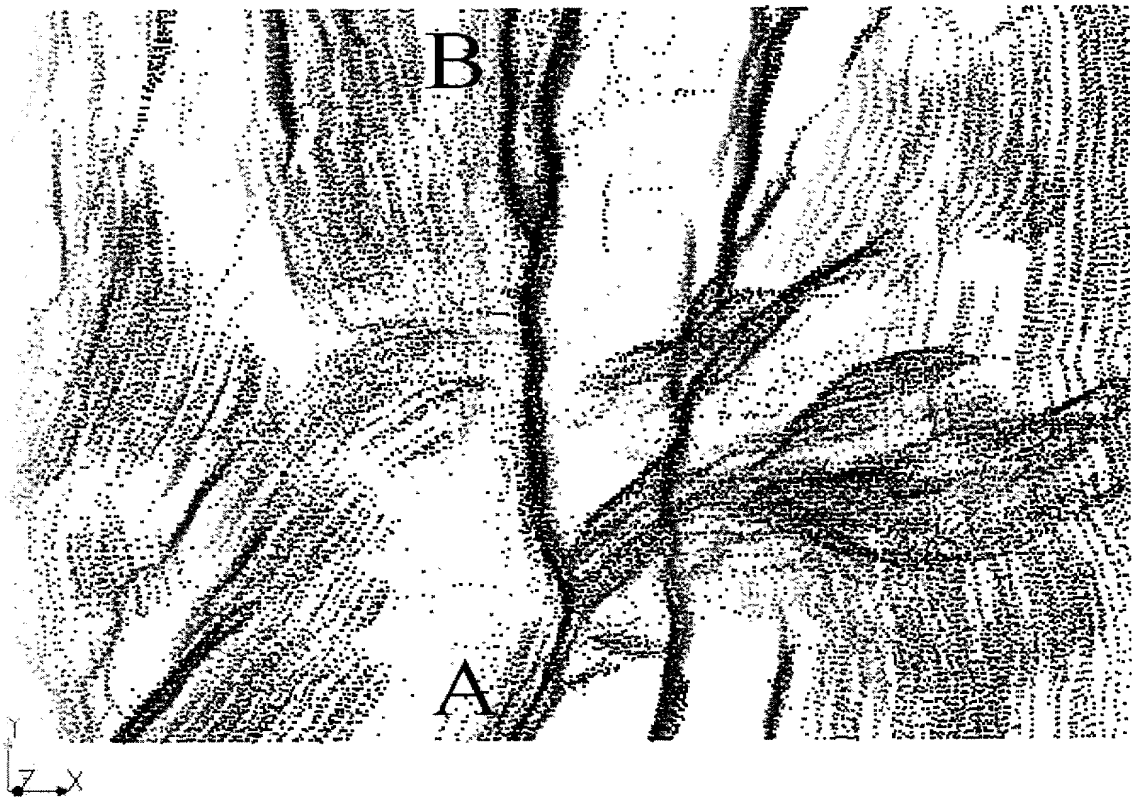


FIG. 21

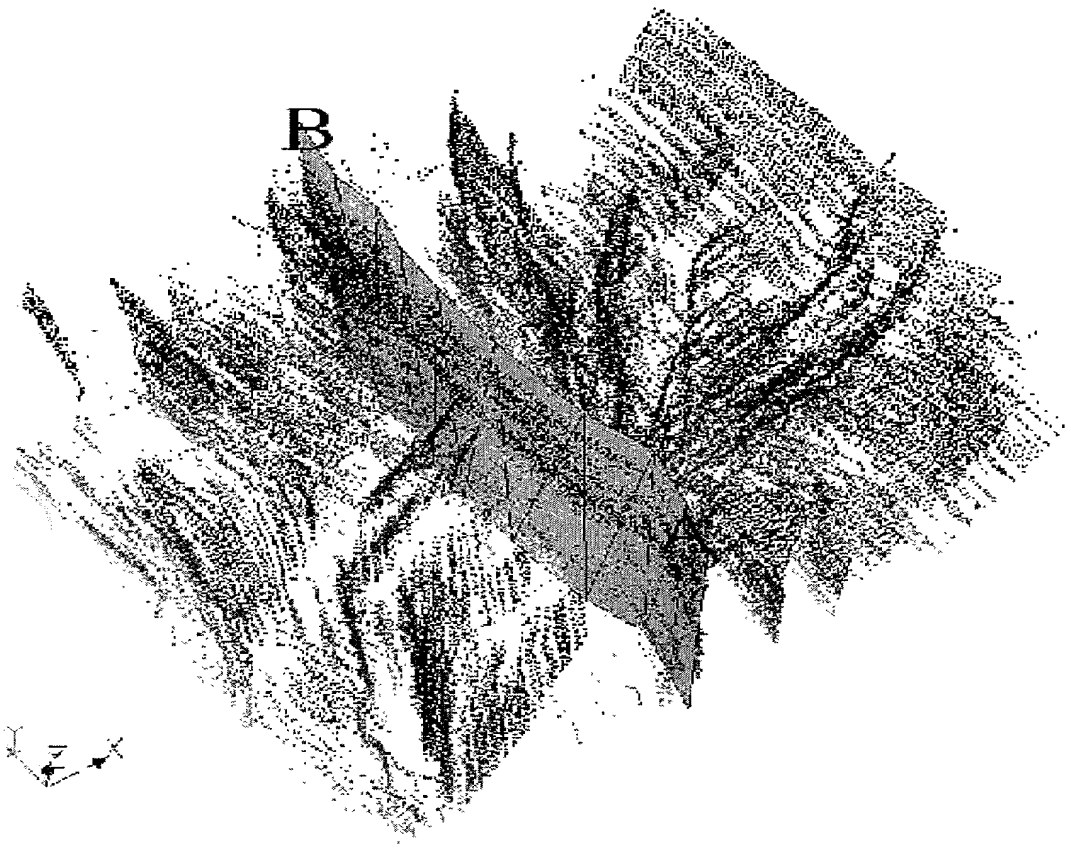


FIG. 22

FIG. 23

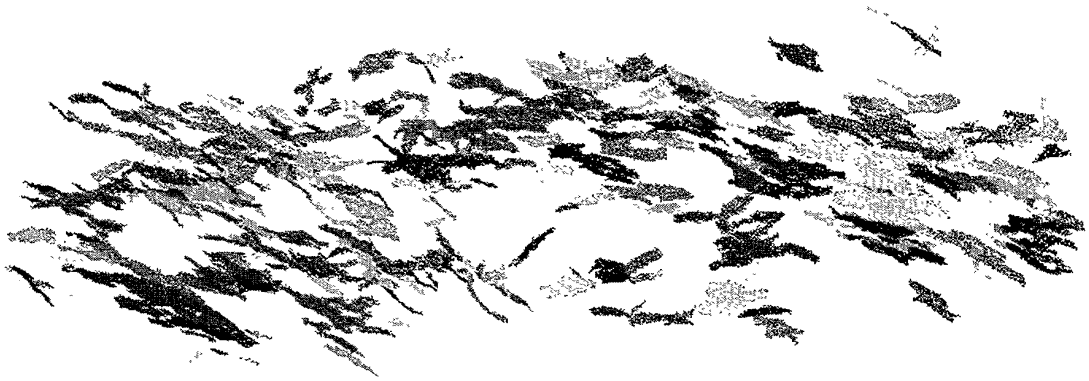


FIG. 24

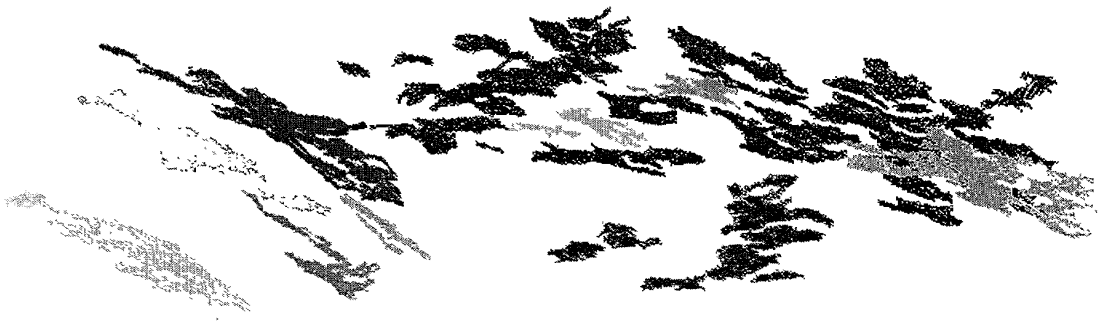


FIG. 25

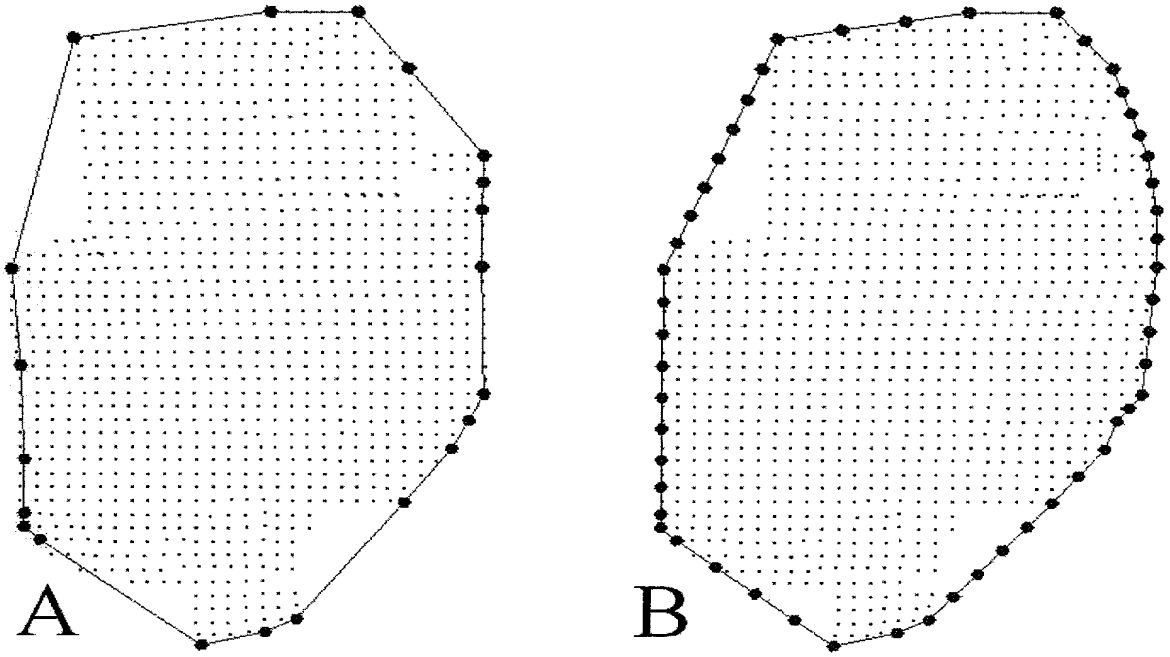


FIG. 26